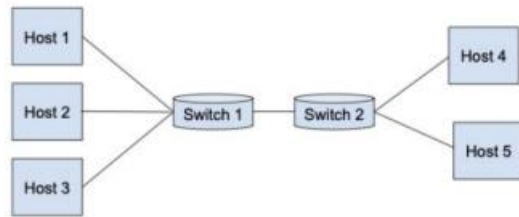


ASSIGNMENT 03

Answers to all the questions are based on the below custom topology



Custom Topology

Controller Hub and Learning Switch

1.1 Hub Controller

The following is the output for the Dumb Hub Controller A simple hub controller forwards packets to all its ports except the incoming one without learning routes. Consequently, it lacks knowledge of past communications, resulting in flooding all ports when it receives requests from the same or new hosts. This behavior leads to consistent and similar response times when conducting multiple pings. And this is what can be verified from the results highlighted below.

A. Controller_hub

```
root@mininet-vm:/home/mininet/A3# python3 topo.py
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time= 25.7 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time= 7.53 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time= 8.01 ms
--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 7.527/13.758/25.738/8.472 ms
root@mininet-vm:/home/mininet/A3# ryu-manager controller_hub.py
loading app controller_hub.py
loading app ryu.controller.ofp_handler
instantiating app controller_hub.py of LearningSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
```

```

packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3

```

b. Learning Switch

The learning switch operates by observing past network traffic and subsequently constructing a forwarding table based on these observations. This table enables the switch to efficiently handle incoming requests. When a request for a specific destination is received, the switch checks its forwarding table to determine if it has prior knowledge of the destination. If it does, it directly forwards the packet to the appropriate port. If not, it resorts to a flood mode, broadcasting the packet to all outgoing ports. This learning mechanism results in expedited processing for subsequent packets destined for the same location that the switch has previously encountered. This efficiency can be corroborated by analyzing the recorded results. For instance, the initial request may take 26.3 milliseconds to process, while subsequent requests show significantly reduced processing times, such as 0.415 milliseconds and 0.152 milliseconds for the second and third requests, respectively. This demonstrates the switch's ability to remember and respond faster to destinations it has encountered in the past.

```

root@mininet-vm:/home/mininet/A3# python3 topo.py
mininet> h2 ping -c 3 h5
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=26.3 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=0.415 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=0.152 ms
--- 10.0.0.5 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 0.152/8.950/26.285/12.257 ms
root@mininet-vm:/home/mininet/A3# ryu-manager learning_switch.py
loading app learning_switch.py
loading app ryu.controller.ofp_handler
instantiating app learning_switch.py of LearningSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:05 3

```

c. Difference between Hub Controller and Learning Switch

i. Hub Controller: A hub controller, often called a hub or a basic hub, sends incoming packets to all of its ports except the one they originated from. Essentially, it broadcasts the packet to all connected devices, leading to network congestion and inefficiency.

Learning Switch: A learning switch, also referred to as a switch, uses an intelligent packet forwarding mechanism. It observes previous network traffic and creates a forwarding table that links MAC addresses to their corresponding switch ports. When a packet arrives, it checks the forwarding table and directs the packet only to the port where the destination MAC address resides, significantly reducing network traffic and improving efficiency.

ii. Hub Controller: A hub controller lacks the ability to learn. It simply forwards packets to all ports without any knowledge of the connected devices or their MAC addresses.

Learning Switch: A learning switch actively learns the MAC addresses of devices connected to its ports and stores this information in its forwarding table, allowing it to make informed decisions about packet forwarding based on destination MAC addresses.

1.2 Throughput

a. Controller hub

```
root@mininet-vm:/home/mininet# iperf -s
```

```
-----  
Server listening on TCP port 5001  
TCP window size: 85.3 KByte (default)
```

```
-----  
[ 6] local 10.0.0.1 port 5001 connected with 10.0.0.5 port 53814  
[ ID] Interval Transfer Bandwidth  
[ 6] 0.0-13.2 sec 7.12 MBytes 4.53 Mbits/sec  
root@mininet-vm:/home/mininet# iperf -c 10.0.0.1
```

```
-----  
Client connecting to 10.0.0.1, TCP port 5001  
TCP window size: 85.3 KByte (default)
```

```
-----  
[ 5] local 10.0.0.5 port 53814 connected with 10.0.0.1 port 5001  
[ ID] Interval Transfer Bandwidth  
[ 5] 0.0-10.3 sec 7.12 MBytes 5.80 Mbits/sec  
root@mininet-vm:/home/mininet#
```

b. Learning switch

```
root@mininet-vm:/home/mininet# iperf -s
```

```
-----  
Server listening on TCP port 5001  
TCP window size: 85.3 KByte (default)
```

```
-----  
[ 6] local 10.0.0.1 port 5001 connected with 10.0.0.5 port 53830  
[ ID] Interval Transfer Bandwidth
```

```
[ 6] 0.0-10.0 sec 26.0 GBytes 22.3 Gbits/sec
root@mininet-vm:/home/mininet# iperf -c 10.0.0.1
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
[ 5] local 10.0.0.5 port 53830 connected with 10.0.0.1 port 5001
[ ID] Interval Transfer Bandwidth
[ 5] 0.0-10.0 sec 26.0 GBytes 22.3 Gbits/sec
root@mininet-vm:/home/mininet#
```

Here we can verify that the throughput for the learning switch is better than the controller hub due to less redundant flooding messages as the learning switch maintains the forwarding table.

1.3 The following are the installed rules on switches S1 and S2

- a. Controller hub** : This is dumb switch and does not store any information and can be verified from the below output.

```
root@mininet-vm:/home/mininet/A3# python3 topo.py
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> dpctl dump-flows
*** s1 -----
cookie=0x0, duration=27.278s, table=0, n_packets=80, n_bytes=5600, priority=0
actions=CONTROLLER:65535
*** s2 -----
cookie=0x0, duration=27.287s, table=0, n_packets=80, n_bytes=5600, priority=0
actions=CONTROLLER:65535
```

- b. Learning switch** : Store rules in the forwarding tables.

```
root@mininet-vm:/home/mininet/A3# python3 topo.py
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> dpctl dump-flows
```

```

*** s1 -----
cookie=0x0, duration=7.748s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
cookie=0x0, duration=7.741s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
cookie=0x0, duration=7.723s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
cookie=0x0, duration=7.717s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
cookie=0x0, duration=7.690s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
cookie=0x0, duration=7.688s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"
cookie=0x0, duration=7.661s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"
cookie=0x0, duration=7.657s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=output:"s1-eth4"
cookie=0x0, duration=7.629s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
cookie=0x0, duration=7.627s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
cookie=0x0, duration=7.600s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
cookie=0x0, duration=7.595s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"
cookie=0x0, duration=7.573s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"
cookie=0x0, duration=7.569s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:05
actions=output:"s1-eth4"
cookie=0x0, duration=7.528s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
cookie=0x0, duration=7.523s, table=0, n_packets=2, n_bytes=140,

```

priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"
cookie=0x0, duration=7.501s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
cookie=0x0, duration=7.498s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05
actions=output:"s1-eth4"
cookie=0x0, duration=11.146s, table=0, n_packets=28, n_bytes=1680, priority=0
actions=CONTROLLER:65535

*** s2 -----

cookie=0x0, duration=7.703s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:"s2-eth3"
cookie=0x0, duration=7.696s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:04
actions=output:"s2-eth1"
cookie=0x0, duration=7.672s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=output:"s2-eth3"
cookie=0x0, duration=7.663s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=output:"s2-eth2"
cookie=0x0, duration=7.617s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=output:"s2-eth3"
cookie=0x0, duration=7.601s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=output:"s2-eth1"
cookie=0x0, duration=7.586s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:02
actions=output:"s2-eth3"
cookie=0x0, duration=7.576s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:05
actions=output:"s2-eth2"
cookie=0x0, duration=7.548s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03
actions=output:"s2-eth3"
cookie=0x0, duration=7.530s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04
actions=output:"s2-eth1"
cookie=0x0, duration=7.514s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03
actions=output:"s2-eth3"
cookie=0x0, duration=7.504s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05
actions=output:"s2-eth2"

```

cookie=0x0, duration=7.466s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:04
actions=output:"s2-eth1"
cookie=0x0, duration=7.457s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:05
actions=output:"s2-eth2"
cookie=0x0, duration=11.153s, table=0, n_packets=24, n_bytes=1400, priority=0
actions=CONTROLLER:65535

```

2 Firewall and Monitor

2.1 In this context, our objective is to establish restrictions on communication between certain hosts while also monitoring incoming packets from host H3 on the switch. This can be confirmed through the results obtained when performing a 'pingall' operation, which involves sending ping requests from all hosts to all other hosts. Specifically, we aim to prevent communication between H2 and H3, as well as between H5 and H1 with H4. These restrictions are evident in the results, where attempts to ping between these host pairs are indicated with an 'X,' signifying the inability to establish a connection. Furthermore, during this evaluation, we keep a tally of all incoming packets originating from host H3, which allows us to monitor and track the communication patterns of this particular host on the switch.

- a. Prevent communication between H2 and H3 with H5, and H1 with H4

```
root@mininet-vm:/home/mininet/A3# python3 topo.py
```

```
mininet> pingall
```

```
*** Ping: testing ping reachability
```

```
h1 -> h2 h3 X h5
```

```
h2 -> h1 h3 h4 X
```

```
h3 -> h1 h2 h4 X
```

```
h4 -> X h2 h3 h5
```

```
h5 -> h1 X X h4
```

```
*** Results: 30% dropped (14/20 received)
```

- b. Counting the packets for H3 on switch S1

```
root@mininet-vm:/home/mininet/A3# ryu-manager firewall_monitor.py
```

```
loading app firewall_monitor.py
```

```
loading app ryu.controller.ofp_handler
```

```
instantiating app firewall_monitor.py of FirewallApp
```

```
instantiating app ryu.controller.ofp_handler of OFPHandler
```

```
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
```

```
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
```

```
packet in 1 00:00:00:00:00:02 00:00:00:00:00:01 2
```

```
packet in 1 00:00:00:00:00:01 00:00:00:00:00:02 1
```

```
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
```

```
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
```

```
packet in 1 00:00:00:00:00:03 00:00:00:00:00:01 3
```

```
Packet count for h3 on switch s1: 1
```

```
packet in 1 00:00:00:00:00:01 00:00:00:00:00:03 1
```

```
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
```

packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:04 00:00:00:00:00:01 1
packet in 1 00:00:00:00:00:04 00:00:00:00:00:01 4
packet in 1 00:00:00:00:00:01 00:00:00:00:00:04 1
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:01 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:01 4
packet in 1 00:00:00:00:00:01 00:00:00:00:00:05 1
packet in 2 00:00:00:00:00:01 00:00:00:00:00:05 3
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 3
packet in 1 00:00:00:00:00:03 00:00:00:00:00:02 3

Packet count for h3 on switch s1: 2

packet in 1 00:00:00:00:00:02 00:00:00:00:00:03 2
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:04 00:00:00:00:00:02 1
packet in 1 00:00:00:00:00:04 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:04 2
packet in 2 00:00:00:00:00:02 00:00:00:00:00:04 3
packet in 1 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 2
packet in 2 00:00:00:00:00:02 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:02 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:02 4
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 1 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff 3

Packet count for h3 on switch s1: 3

packet in 2 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:04 00:00:00:00:00:03 1
packet in 1 00:00:00:00:00:04 00:00:00:00:00:03 4
packet in 1 00:00:00:00:00:03 00:00:00:00:00:04 3

Packet count for h3 on switch s1: 4

packet in 2 00:00:00:00:00:03 00:00:00:00:00:04 3
packet in 1 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff 3

Packet count for h3 on switch s1: 5

packet in 2 00:00:00:00:00:03 ff:ff:ff:ff:ff:ff 3
packet in 2 00:00:00:00:00:05 00:00:00:00:00:03 2
packet in 1 00:00:00:00:00:05 00:00:00:00:00:03 4
packet in 1 00:00:00:00:00:03 00:00:00:00:00:05 3
packet in 1 00:00:00:00:00:01 00:00:00:00:00:04 1
packet in 1 00:00:00:00:00:01 00:00:00:00:00:04 1
packet in 1 00:00:00:00:00:01 00:00:00:00:00:04 1
packet in 2 00:00:00:00:00:04 ff:ff:ff:ff:ff:ff 1
packet in 1 00:00:00:00:00:04 ff:ff:ff:ff:ff:ff 4
packet in 2 00:00:00:00:00:05 00:00:00:00:00:04 2
packet in 2 00:00:00:00:00:04 00:00:00:00:00:05 1

packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 1 00:00:00:00:00:02 00:00:00:00:00:05 2
packet in 1 00:00:00:00:00:03 00:00:00:00:00:05 3
packet in 1 00:00:00:00:00:03 00:00:00:00:00:05 3
packet in 1 00:00:00:00:00:03 00:00:00:00:00:05 3
packet in 1 00:00:00:00:00:03 00:00:00:00:00:05 3

c. Installed rules on switches S1 and S2

mininet> dpctl dump-flows

*** s1 -----

cookie=0x0, duration=189.253s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"

cookie=0x0, duration=189.246s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"

cookie=0x0, duration=189.229s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"

cookie=0x0, duration=189.223s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"

cookie=0x0, duration=189.201s, table=0, n_packets=4, n_bytes=224,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"

cookie=0x0, duration=179.180s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=output:"s1-eth1"

cookie=0x0, duration=179.174s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth1",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=output:"s1-eth4"

cookie=0x0, duration=179.141s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"

cookie=0x0, duration=179.138s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"

cookie=0x0, duration=179.114s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"

cookie=0x0, duration=179.109s, table=0, n_packets=3, n_bytes=182,
priority=1,in_port="s1-eth2",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"

cookie=0x0, duration=179.092s, table=0, n_packets=4, n_bytes=224,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:02
actions=output:"s1-eth2"

cookie=0x0, duration=169.064s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
cookie=0x0, duration=169.061s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s1-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04
actions=output:"s1-eth4"
cookie=0x0, duration=169.034s, table=0, n_packets=4, n_bytes=224,
priority=1,in_port="s1-eth4",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03
actions=output:"s1-eth3"
cookie=0x0, duration=192.425s, table=0, n_packets=40, n_bytes=2352, priority=0
actions=CONTROLLER:65535

*** s2 -----

cookie=0x0, duration=189.213s, table=0, n_packets=4, n_bytes=224,
priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:01
actions=output:"s2-eth3"
cookie=0x0, duration=179.198s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:01
actions=output:"s2-eth3"
cookie=0x0, duration=179.176s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:05
actions=output:"s2-eth2"
cookie=0x0, duration=179.133s, table=0, n_packets=4, n_bytes=280,
priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:02
actions=output:"s2-eth3"
cookie=0x0, duration=179.115s, table=0, n_packets=3, n_bytes=182,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:04
actions=output:"s2-eth1"
cookie=0x0, duration=179.103s, table=0, n_packets=4, n_bytes=224,
priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:02
actions=output:"s2-eth3"
cookie=0x0, duration=169.078s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:03
actions=output:"s2-eth3"
cookie=0x0, duration=169.067s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth3",dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:04
actions=output:"s2-eth1"
cookie=0x0, duration=169.052s, table=0, n_packets=4, n_bytes=224,
priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03
actions=output:"s2-eth3"
cookie=0x0, duration=149.012s, table=0, n_packets=3, n_bytes=238,
priority=1,in_port="s2-eth2",dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:04
actions=output:"s2-eth1"
cookie=0x0, duration=149.010s, table=0, n_packets=2, n_bytes=140,
priority=1,in_port="s2-eth1",dl_src=00:00:00:00:00:04,dl_dst=00:00:00:00:00:05
actions=output:"s2-eth2"
cookie=0x0, duration=192.431s, table=0, n_packets=21, n_bytes=1106, priority=0
actions=CONTROLLER:65535

2.2. Can you suggest strategies to reduce the number of firewall rules on the switch?

- a. Grouping Hosts: Instead of defining rules for individual hosts, cluster hosts into subnets and enforce rules at the subnet level. For example, if hosts H2, H3, and H5 belong to the same subnet, you can create a single rule to control communication to or from the entire subnet.
- b. Utilizing Wildcard Rules: Employ wildcard rules or wildcard masks to represent multiple IP addresses or subnets with a single rule. For instance, you can establish a rule to manage traffic from any IP within a specific range.
- c. Port or Service Focus: Instead of specifying rules for individual hosts, concentrate on regulating traffic by port or service. This approach is especially beneficial when you want to restrict access to particular services regardless of the source or destination.

2.3 Implementing Real-Time Firewall Policies without Disruption

- a. Employ Dynamic Firewall Rules: Implement firewall rules in a dynamic fashion, allowing for real-time updates. This involves using a controller or firewall management system capable of pushing updates to switches automatically, without manual intervention. The new rules will either override or complement the existing ones as required.
- b. Prioritize Real-Time Policies: When implementing real-time policies, ensure that these policies take precedence over the existing rules. This can be achieved by establishing rule priorities, giving higher priority to real-time rules, which means they are processed before the pre-existing rules.
- c. Maintain Rule Version Control: When applying real-time policies, label them with a version number or timestamp. This way, you can easily monitor and manage the progression of your rules and revert to previous versions if necessary.

3 Load Balancer

3.1 In this portion of the assignment, our task was to implement a load balancer to evenly distribute the workload between servers H4 and H5. To accomplish this, we devised a method to switch incoming requests between these servers in an alternating manner. Consequently, when a new packet request arrives at switch S1, it is sent to either host H4 or H5 in a rotating sequence, effectively distributing the workload and achieving an equitable utilization of both servers.

The results presented below serve as confirmation of the anticipated outcomes. For instance, when initiating a ping request from host H1, we can observe that it is initially directed to server 10.0.0.4, and upon subsequent requests, it is routed to server 10.0.0.5. This illustrates the effectiveness of the load balancer in achieving the desired workload distribution, as requests are dynamically directed to the servers in an alternating fashion.

- a. Mininet ping to the VIRTUAL_IP(10.0.0.42)

```

root@mininet-vm:/home/mininet/A3# python3 topo.py
mininet> h1 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.
--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
mininet> h1 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.
--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
mininet> h2 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.
--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms
mininet> h2 ping -c 1 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.
--- 10.0.0.42 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

```

b. Selected servers for requests

```

root@mininet-vm:/home/mininet/A3# nano load_balancer.py
root@mininet-vm:/home/mininet/A3# ryu-manager load_balancer.py
loading app load_balancer.py
loading app ryu.controller.ofp_handler
instantiating app load_balancer.py of SimpleSwitch13
Done with initial setup related to server list creation.
instantiating app ryu.controller.ofp_handler of OFPHandler
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
:::Sent the packet_out:::
The selected server is ==> 10.0.0.4
<=====Packet from client: 10.0.0.1. Sent to server: 10.0.0.4, MAC: 00:00:00:00:00:04
and on switch port: 1=====>
<+++++++Reply sent from server: 10.0.0.4, MAC: 00:00:00:00:00:04. Via load balancer:
10.0.0.42. To client: 10.0.0.1+++++++>
The selected server is ==> 10.0.0.5
<=====Packet from client: 10.0.0.1. Sent to server: 10.0.0.5, MAC: 00:00:00:00:00:05
and on switch port: 2=====>
<+++++++Reply sent from server: 10.0.0.5, MAC: 00:00:00:00:00:05. Via load balancer:
10.0.0.42. To client: 10.0.0.1+++++++>
(((Entered the ARP Reply function to build a packet and reply back appropriately)))
{{{Exiting the ARP Reply Function as done with processing for ARP reply packet}}}
:::Sent the packet_out:::
The selected server is ==> 10.0.0.4
<=====Packet from client: 10.0.0.2. Sent to server: 10.0.0.4, MAC: 00:00:00:00:00:04
and on switch port: 1=====>
<+++++++Reply sent from server: 10.0.0.4, MAC: 00:00:00:00:00:04. Via load balancer:
10.0.0.42. To client: 10.0.0.2+++++++>
The selected server is ==> 10.0.0.5

```

<=====Packet from client: 10.0.0.2. Sent to server: 10.0.0.5, MAC: 00:00:00:00:00:05
and on switch port: 2=====>
<+++++++Reply sent from server: 10.0.0.5, MAC: 00:00:00:00:00:05. Via load balancer:
10.0.0.42. To client: 10.0.0.2+++++++>

c. (H1, H2, and H3) ping the virtual IP and the installed rule on the switches

```
mininet> h1 ping -c 3 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.
--- 10.0.0.42 ping statistics ---
3 packets transmitted, 100% received, 0 packet loss, time 2041ms
mininet> h2 ping -c 3 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.
--- 10.0.0.42 ping statistics ---
3 packets transmitted, 100% received, 0 packet loss, time 2033ms
mininet> h3 ping -c 3 10.0.0.42
PING 10.0.0.42 (10.0.0.42) 56(84) bytes of data.
--- 10.0.0.42 ping statistics ---
3 packets transmitted, 100% received, 0 packet loss, time 2053ms
mininet> dpctl dump-flows
*** s1 -----
cookie=0x0, duration=279.401s, table=0, n_packets=6, n_bytes=420, priority=0
actions=CONTROLLER:65535
*** s2 -----
cookie=0x0, duration=279.407s, table=0, n_packets=0, n_bytes=0, priority=0
actions=CONTROLLER:65535
mininet>
```

3.2. If you were to implement a load balancing policy that considers the load on these servers, what additional steps would you take?

- a. Implement a mechanism to monitor the health and load of the servers (H4 and H5). This can involve periodically sending health checks or pings to the servers to assess their responsiveness and availability.
- b. Develop logic to dynamically adjust the load balancing strategy based on the current server loads. This might include collecting statistics on server response times, CPU usage, or other relevant metrics. If one server becomes overloaded or unresponsive, the load balancer should direct less traffic to it.
- c. Modify the load balancing algorithm to use weighted round-robin. Assign weights to each server based on their capacity or performance. For example, if H4 is more powerful than H5, it can be assigned a higher weight to receive a larger share of the traffic. This allows you to distribute the load in a way that is proportional to the servers' capabilities.